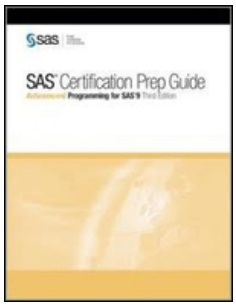


SAS Certification Prep Guide: Advanced Programming for SAS 9, Third Edition

by SAS Institute

SAS Institute. (c) 2011. Copying Prohibited.

Reprinted for Madhusmita Nayak, Accenture

madhusmita.nayak@accenture.com

Reprinted with permission as a subscription benefit of **Skillport**,
http://skillport.books24x7.com/

All rights reserved. Reproduction and/or distribution in whole or in part in electronic,paper or other forms without written permission is prohibited.

Chapter 23: Selecting Efficient Sorting Strategies

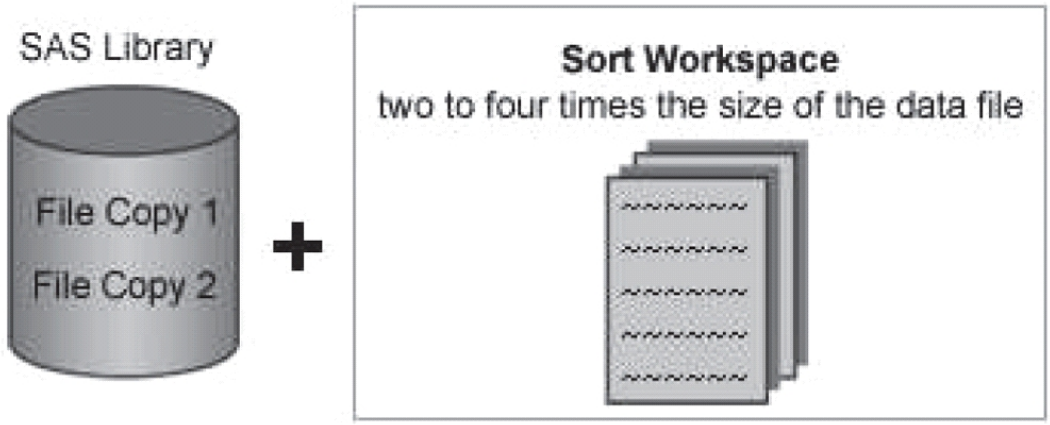
Overview

Introduction

Sometimes you need to group observations by the values of aparticular variable or order the observations in a particular way, such as alphabetically, in order to

- reorder the data for reporting
- reduce data retrieval time
- enable BY-group processing in both DATA and PROC steps.

The SORT procedure is one technique that can be used to group or order data. However, the SORT procedure can use a high volume of resources. When using PROC SORT to sort an uncompressed data file and replace it with the sorted version, SAS requires enough space in the data library to hold two copies of the data file, plus a workspace that is approximately two to four times the size of the data file.



In some cases, you might be able to use techniques other than the SORT procedure to group or order observations. In other cases, you might be able to use options or techniques with the SORT procedure that enable you to minimize resource usage.

Note This chapter does not cover the SAS Scalable Performance Data Engine (SAS SPD Engine). For details about using the SAS SPD Engine to improve performance, see the SAS documentation.

Objectives

In this chapter, you learn to

- apply techniques that enable you to avoid unnecessary sorts
- calculate and allocate sort resources
- use strategies for sorting large data sets
- eliminate duplicate observations efficiently.

Prerequisites

Before beginning this chapter, you should complete the following chapters:

- Part 1: SQL Processing with SAS

Reprinted for OET7P/639280, Accenture
SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

Avoiding Unnecessary Sorts

Overview

In some cases you can avoid a sort by using

- BY-group processing with an index
- BY-group processing with the NOTSORTED option
- a CLASS statement
- the SORTEDBY= data set option.

Using BY-Group Processing with an Index

BY-group processing is a method of processing observations from one or more SAS data sets that are grouped or ordered by the values of one or more common variables. You can use BY-group processing in both DATA steps and PROC steps.

General form, BY statement:

BY *variable(s);*

where

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

variable(s)

names each variable by which the data set is sorted or indexed.

The most common use of BY-group processing in the DATA step is to combine two or more SAS data sets by using the BY statement with a SET, MERGE, UPDATE, or MODIFY statement. When you use a BY statement with a SET, MERGE, or UPDATE statement, the data sets must first be ordered on the values of the BY variable unless you index the data sets. You can also use the NOTSORTED option in the BY statement with a SET statement.

When BY-group processing is used with an index that is based on one of the BY variables, the data can be sequenced without using the SORT procedure. The data can be sequenced by different variables if multiple indexes are used. Because indexes are updated automatically, there is no need to re-sort a data set when observations are modified or added.

However, using BY-group processing with an index has two disadvantages:

- It is generally less efficient than sequentially reading a sorted data set because processing BY groups typically means retrieving the entire file.
- It requires storage space for the index.

Note A BY statement does not use an index if the BY statement includes the DESCENDING or NOTSORTED option or if SAS detects that the data file is physically stored in sorted order on the BY variables.

Note If you use a MODIFY statement, the data does not need to be ordered. However, your program might run more efficiently with ordered data.

Comparative Example: Using BY-Group Processing with an Index to Avoid a Sort

Overview

Suppose you want to use an existing data set, *Retail.Order_fact*, to create a new SAS data set that is ordered by the variable **`Order_Date`**. You could accomplish this task using

1. BY-Group Processing with an Index, Data in Random Order
2. Presorted Data in a DATA Step
3. PROC SORT Followed by a DATA Step.

The following sample programs show each of these techniques. You can use these samples as models for creating benchmark programs in your own environment. Your results might vary depending on the structure of your data, your operating environment, and the resources that are available at your site. You can also view general recommendations for BY-group processing with an index or sort.

Programming Techniques

---

❶ BY-Group Processing with an Index, Data in Random Order

In this example, the SAS data set *Retail.Order_fact* is indexed on the variable **`Order_Date`**. The data in *Retail.Order_fact* is in random order.

```
data _null_;
   set retail.order_fact;
   by order_date;
run;
```

---

❷ Presorted Data in a DATA Step

In this example, the SAS data set *Retail.Order_fact* is sorted on the variable **`Order_Date`** before it is read using the

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

DATA step.

```
data _null_;
   set retail.order_fact;
   by order_date;
run;
```

---

❸ PROC SORT Followed by a DATA STEP

In this example, the SAS data set *Retail.Order_fact* is sorted using the SORT procedure. The data is then read using the DATA step.

```
proc sort data=retail.order_fact;
   by order_date;
run;
data _null_;
   set retail.order_fact;
   by order_date;
run;
```

General Recommendations

- To conserve resources, use sort order rather than an index for BY-group processing.
- Although using an index for BY-group processing is less efficient than using sort order, it might be the best choice if resource limitations make sorting a file difficult.

Using the NOTSORTED Option

You can also use the *NOTSORTED option* with a BY statement to create ordered or grouped reports without sorting the data. The NOTSORTED option specifies that observations that have the same BY value are grouped together but are not necessarily sorted in alphabetical or numeric order.

General form, BY statement with the NOTSORTED option:

BY *variable(s)* **NOTSORTED;**

where

variable(s)

names each variable by which the data set is sorted or indexed.

The NOTSORTED option can appear anywhere in the BY statement and is useful if you have data that is in logical categories or groupings such as chronological order. The NOTSORTED option works best when observations that have the same BY value are stored together.

Caution The NOTSORTED option turns off sequence checking. If your data is not grouped, using the NOTSORTED option can produce a large amount of output.

Caution The NOTSORTED option cannot be used with the MERGE or UPDATE statements.

Example

Suppose you want to use the PRINT procedure to print the contents of the data set *Retail.Europe*, which contains data about European customers. The data set includes values for country names **(`Country_Name`)** as well as two-character country codes **(`Country_Code`)**.

The data is grouped and sorted by the values of **`Country_Code`**. However, you want the observations in the output to be

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

grouped by the values of **Country_Name**.

Table 23.1: Country Codes and Country Names

Two-Character Country Code	Full-Text Country Name
BE	Belgium
DE	Germany
DK	Denmark
ES	Spain
FI	Finland
FR	France
GB	United Kingdom
GR	Greece
IE	Ireland
IT	Italy
LU	Luxembourg
NL	Netherlands

Table 23.2: SAS Data Set Retail.Europe, Selected Observations

Obs	Customer_ID	Customer_Country	Customer_Name	Country
374	93803	BE	Luc Vandeloo	Belgium
375	93980	BE	Saida Van Itterbeeck	Belgium
376	94040	BE	Pat Sodergard	Belgium
377	13	DE	Markus Sepke	Germany
378	19	DE	Oliver S. Füßling	Germany
379	50	DE	Gert-Gunter Mendier	Germany

You can use the NOTSORTED option with a BY statement to accomplish this task without using the SORT procedure.

```
proc print data=retail.europe;
   by country notsorted;
run;
```

PROC PRINT output shows that the data is grouped but is not in sorted order. For example, observations in which the value of **Country_Name** is *Germany* are followed by observations in which the value of **Country_Name** is *Denmark*.

Table 23.3: PROC PRINT Output, Selected Observations

Country_Name=Germany			
Obs	Customer_ID	Customer_Name	Country_Code
2340	65751	Dieter Krüger	DE
2341	65759	Fredy Ensinger	DE
2342	65772	Dieter Wein	DE
2343	65784	Claudia Martin	DE
2344	65793	York Bückel	DE

Country_Name=Denmark			
Obs	Customer_ID	Customer_Name	Country_Code
2345	1331	Else Gade Jensen	DK
2346	1531	Maria Bargmann Hersom	DK

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

2347	1751	Flemming Schønberg Schleiss	DK
2348	2175	Marlene Lœrke Shah	DK
2349	2486	Lars Flensted-Jensen	DK

Using FIRST. and LAST.

The NOTSORTED option can be used with **FIRST**. *variable* and **LAST**. *variable*, which are temporary automatic variables in the PDV that identify the first and last observations in each BY group.

These temporary variables are available for DATA step programming but are not added to the output data set. Their values indicate whether an observation is

- the first one in a BY group
- the last one in a BY group
- neither the first nor the last one in a BY group
- both first and last, as is the case when there is only one observation in a BY group.

You can take actions conditionally, based on whether you are processing the first observation of a BY group or the last.

When an observation is the first in a BY group, SAS sets the value of **FIRST.*variable*** to *1*. For all other observations in the BY group, the value of **FIRST.*variable*** is *0*. Likewise, if an observation is the last in a BY group, SAS sets the value of **LAST.*variable*** to *1*. For all other observations in the BY group, the value of **LAST.*variable*** is *0*.

Example

The following program creates a new SAS data set *Work.New*. In the input data set, observations that have the same value for **Ordername** *(Retail, Catalog*, or *Internet)* are grouped together.

```
data work.new;
   set company.sales;
   by ordername notsorted;
   if first.ordername;
run;
```

When the program is submitted, SAS creates the temporary variables **FIRST.Ordername** and **LAST.Ordername**. These variables can be used during the DATA step, but they do not become variables in the new data set. The value 1 flags the beginning and end of each value in the BY group:

Observations			Corresponding FIRST. and LAST. Values	
Obs	Customer_ID	OrderName	FIRST.OrderName	LAST.OrderName
1*	11791	Retail	1	0
2	8406	Retail	0	0
3	71020	Retail	0	0
4	21735	Retail	0	1
5*	82141	Catalog	1	0
6	30993	Catalog	0	1
7*	579	Internet	1	0
8	77184	Internet	0	1

* This observation is output to the data set *work.new*.

Using the GROUPFORMAT Option

The GROUPFORMAT option uses the formatted values of avariable instead of the internal values to determine where a BY group begins and ends, and how **FIRST.*variable*** and **LAST.*variable*** are computed.

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

General form, BY statement with the GROUPFORMAT option:

BY *variable(s)* **GROUPFORMAT;**

where

variable(s)

names each variable by which the data set is sorted or indexed.

The GROUPFORMAT option

- is available only in the DATA step
- is useful when you define formats for grouped data
- enables the DATA step to process the same groups of data as a summary procedure or PROC REPORT.

When the GROUPFORMAT option is used, the data set must be sorted by the GROUPFORMAT variable or grouped by the formatted values of the GROUPFORMAT variable.

Example

Suppose you want to create a summary report that includes the number of orders for each quarter in 2002. The data for the report is stored in the SAS data set *Company.Orders*.

Table 23.4: SAS Data Set Company.Orders, First Five Observations

Obs	Order_ID	Order_Type	Employee_ID	Customer_ID	Order_Date	Delivery_Date
1	1230000033	Internet Sale	99999999	8818	01JAN1998	07JAN1998
2	1230000204	Internet Sale	99999999	47793	01JAN1998	04JAN1998
3	1230000268	Internet Sale	99999999	71727	01JAN1998	03JAN1998
4	1230000487	Internet Sale	99999999	74503	01JAN1998	04JAN1998
5	1230000494	Internet Sale	99999999	8610	01JAN1998	07JAN1998

By creating a format for the data and using the GROUPFORMAT and NOTSORTED options, you cause SAS to create the variables **FIRST.Order_Date** and **LAST.Order_Date** based on the formatted values, not the internal values. This groups the data without requiring the creation of a new variable.

```
proc format;
   value qtrfmt '01jan2002'd - '31mar2002'd = '1'
                '01apr2002'd - '30jun2002'd = '2'
                '01jul2002'd - '30sep2002'd = '3'
                '01oct2002'd - '31dec2002'd = '4';
run;

data company.quarters(keep=count order_date
           rename=(order_date=Quarter));
   set company.orders;
   format order_date qtrfmt.;
   by order_date groupformat notsorted;
   where year(order_date)=2002;
   if first.order_date then Count=0;
   Count +1;
   if last.order_date;
run;
```

SAS Data Set Company.Quarters

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

Obs	Quarter	Count
1	1	4545
2	2	5330
3	3	5508
4	4	5649

Using the CLASS Statement

You can also use a *CLASS statement* to avoid a sort. Unlike the BY statement, the CLASS statement does not require the data to be presorted using the CLASS values, nor does it require an index that is based on the CLASS variables.

If the data cannot be sorted, the CLASS statement can be used. However, unlike using the BY statement, presorting the data for use with a CLASS statement does not provide a significant benefit.

General form, CLASS statement:

CLASS *variable(s) </options>;*

where

variable(s)

specifies one or more variables that the procedure uses to group the data.

Remember that a CLASS statement specifies the variables whose values define the subgroup combinations for an analysis by a SAS procedure. You can use the CLASS statement with the following Base SAS procedures:

- MEANS
- TABULATE
- SUMMARY
- UNIVARIATE.

Variables in a CLASS statement are referred to as *class variables*. Class variables can be numeric or character. Class variables can have continuous values, but they typically have a few discrete values that define the classifications of the variable.

Caution The comparison of the use of CLASS and BY statements is appropriate for Base SAS procedures only.

Example

The *Company.Orders* data set contains the variable **`Order_Type`**, which has three discrete values:

- *Retail*
- *Catalog*
- *Internet*.

Suppose you want to show the average retail price, cost per unit, and discount for each value of **`Order_Type`**. You could use the MEANS procedure with either a BY statement or a CLASS statement to complete this task. The statistics created with either of these techniques are the same. However, the report layouts differ.

When the BY statement is used, SAS creates a report for each value of the BY variable. The statistics for each value of **`Order_Type`** appear in a separate tabular report.

```
proc sort data=company.order_fact(keep=order_type
```


Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

```
          quantity total_retail_price
          costprice_per_unit discount) out=company.orders;
   by order_type;
run;

proc means data=company.orders mean;
   by order_type;
   var total_retail_price -- discount;
   freq quantity;
run;
```

Table 23.5: Output, PROC MEANS with a BY Statement

Order Type=Retail Sale		
Variable	Label	Mean
Total-Retail_Price	Total Retail Price for This Product	177.0073169
CostPrice_Per_Unit	Cost Price Per Unit	38.3716413
Discount	Discount in percent of Normal Total Retail Price	0.3881087

Order Type=Catalog Sale		
Variable	Label	Mean
Total-Retail_Price	Total Retail Price for This Product	196.3417829
CostPrice_Per_Unit	Cost Price Per Unit	40.8664533
Discount	Discount in percent of Normal Total Retail Price	0.3885046

Order Type=Internet Sale		
Variable	Label	Mean
Total-Retail_Price	Total Retail Price for This Product	200.2136179
CostPrice_Per_Unit	Cost Price Per Unit	41.6397101
Discount	Discount in percent of Normal Total Retail Price	0.3945766

When the CLASS statement is used, only one report is created. The statistics for each value of **`Order_Type`** are consolidated into one tabular report.

```
proc means
     data=company.orders(keep=order_type
          quantity total_retail_price
          costprice_per_unit discount) mean;
     class order_type;
     var total_retail_price -- discount;
     freq quantity;
run;
```

Output, PROC MEANS with a CLASS Statement			
Order_Type	**N Obs**	**Variable**	**Mean**
Retail Sale	1184633	Total_Retail_Price	177.0073169
		CostPrice_Per_Unit	38.3716413
		Discount	0.3881087
Catalog Sale	222195	Total_Retail_Price	196.3417829
		CostPrice_Per_Unit	40.8664533
		Discount	0.3885046
Internet Sale	190489	Total_Retail_Price	200.2136179
		CostPrice_Per_Unit	41.6397101

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

		Discount	0.3945766

Comparative Example: Using a BY or CLASS Statement to Avoid a Sort

Overview

Suppose you want to create a summary report that shows the average retail price, cost per unit, and discount for each type of order in the *Retail.OrderJ'act* data set. Among the techniques you could use are

1. PROC MEANS with a BY Statement, Presorted
2. PROC MEANS with a CLASS Statement
3. PROC MEANS with a CLASS Statement, Presorted
4. PROC SORT and PROC MEANS with a BY Statement.

The following sample programs show each of these techniques. You can use these samples as models for creating benchmark programs in your own environment. Your results might vary depending on the structure of your data, your operating environment, and the resources that are available at your site. You can also view general recommendations for using a BY or CLASS statement to avoid a sort.

Programming Techniques

---

❶ PROC MEANS with a BY Statement, Presorted

The following program creates a report for each value of the BY variable **Order_Type**. Each report contains the mean value for the variables **Total_Retail_Price**, **CostPrice_Per_Unit**, and **Discount**. The input data is presorted by the value of **Order_Type**.

```
proc means data=retail.orders mean;
   by order_type;
   var total_retail_price costprice_per_unit discount;
   freq quantity;
run;
```

---

❷ PROC MEANS with a CLASS Statement

The following program uses a CLASS statement to create a single, tabular report that includes the meanvalue of **Total_Retail_Price**, **CostPrice_Per_Unit**, and **Discount** for each category of **Order_Type**.

```
proc means data=retail.order_fact(keep=order_type
      quantity total_retail_price
      costprice_per_unit discount) mean;
   class order_type;
   var total_retail_price costprice_per_unit discount;
   freq quantity;
run;
```

---

❸ PROC MEANS with a CLASS Statement, Presorted

In the following program, the input data set *Retail.Order_fact* is presorted by the value of **Order_Type**. The PROC MEANS step uses a CLASS statement to create a single, tabular report that includes the mean value of **Total_Retail_Price**, **CostPrice_Per_Unit**, and **Discount** for each category of **Order_Type**.

```
proc means data=retail.orders(keep=order_type
      quantity total_retail_price
      costprice_per_unit discount) mean;
   class order_type;
   var total_retail_price costprice_per_unit discount;
```


Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

```
   freq quantity;
run;
```

4 PROC SORT and PROC MEANS with a BY Statement

In the following program, the PROC SORT step first sorts the input data set *Retail.Order_fact* by the value of **Order_Type**. The PROC MEANS step then creates a report for each value of **Order_Type**. Each report contains the mean value for the variables **Total_Retail_Price**, **CostPrice_Per_Unit**, and **Discount**.

```
proc sort data=retail.order_fact(keep=order_type
     quantity total_retail_price
     costprice_per_unit discount) out=retail.orders;
     by order_type;
run;

proc means data=retail.orders mean;
   by order_type;
   var total_retail_price costprice_per_unit discount;
   freq quantity;
run;
```

General Recommendations

- If you cannot sort the data, use the CLASS statement.
- Do not presort the data for use with a CLASS statement. Presorting the data does not provide a significant benefit.

Using the SORTEDBY= Data Set Option

If you are working with input data that is already sorted, you can specify how the data is ordered by using the SORTEDBY= data set option.

General form, SORTEDBY= data set option:

SORTEDBY=*by-clause </collate-name>* **|** _**NULL**_

where

by-clause

indicates the data order.

collate-name

names the collating sequence that is used for the sort.

NULL

removes any existing sort information.

Note By default, the collating sequence is that of your operating environment. For details on collating sequences, see the SAS documentation for your operating environment.

Although the SORTEDBY= option does not sort a data set, it sets the value of the *Sorted* flag. It does not set the value of the *Validated* sort flag. (PROC SORT sets the Validated sort flag.) To see the values of these flags *(YES* or *NO)*, use PROC CONTENTS.

```
proc contents data=company.transactions;
run;
```


Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

Data Set Name	COMPANY. TRANSACTIONS	**Observations**	5504
Member Type	DATA	**Variables**	3
Engine	V9	**Indexes**	0
Created	22:55 Friday, May 2, 2004	**Observation Length**	32
Last Modified	22:55 Friday, May 2, 2004	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	YES
Label			
Data Representation	WINDOWS		
Encoding	wlatin1 Western (Windows)		

Short information	
Sortedby	Invoice
Validated	NO
Character Set	ANSI

Figure 23.1: Partial PROC CONTENTS Output

Note Most SAS procedures and subsystems check the order of the data as it is processed unless the Validated sort flag is set on the file.

Example

Suppose you want to create a sorted SAS data set from an external file that contains invoice information. The external file is already sorted by invoice number.

You can use the SORTEDBY= data set option to indicate the data are sorted by the value of **`Invoice`**.

```
data company.transactions (sortedby=invoice);
   infile extdata;
   input Invoice 1-4 Item $6-20 Amount comma 6.;
run;
```

When the *Company.Transactions* data set is created, the sort information is stored with it. PROC SORT checks the sort information before it sorts a data set so that data is not re-sorted unnecessarily. If you attempt to re-sort the data, the log indicates that the data set is already sorted and that no additional sorting occurred.

```
proc sort data=company.transactions;
   by invoice;
run;
```

Table 23.6: SAS Log

```
667  proc sort data=work.transactions;
668  by invoice;
669  run;

NOTE: Input data set is already sorted, no sorting done.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.07 seconds
      cpu time            0.03 seconds
```

Note You can specify SORTEDBY=_NULL_ to remove the Sorted flag. The Sorted flag is also removed if you change or add any values of the variables by which the data set is sorted.

Using a Threaded Sort

Overview

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

Threaded processing takes advantage of multiple CPUs by executing multiple threads in parallel (parallel processing). Threaded procedures are completed in less real time than if each task were handled sequentially, although the CPU time is generally increased.

Beginning with SAS 9, the SORT procedure can take advantage of threaded processing. A thread is a single, independent flow of control through a program or within a process.

Threaded sorting is enabled or disabled by using the SAS system option THREADS | NOTHREADS or the THREADS | NOTHREADS procedure option.

General form, SORT procedure with the THREADS | NOTHREADS option:

PROC SORT *SAS-data-set-name* **THREADS | NOTHREADS;**

where

SAS-data-set-name

is a valid SAS data set name.

THREADS

enables threaded sorting.

NOTHREADS

disables threaded sorting.

Note The THREAD | NOTHREADS procedure option overrides the value of the SAS system option THREADS | NOTHREADS. For information about the THREADS | NOTHREADS system option, see the SAS documentation.

When a threaded sort is used, the observations in the input data set are divided into equal temporary subsets, based on the number of processors that are allocated to the SORT procedure. Each subset is then sorted on a different processor. The sorted subsets are then interleaved to re-create the sorted version of the input data set.

Using the CPUCOUNT= System Option

The performance of threaded sorting is affected by the value of the CPUCOUNT= system option. CPUCOUNT= specifies the number of processors that thread-enabled applications should assume will be available for concurrent processing. SAS uses this information to determine how many threads to start, not to restrict the number of CPUs that will be used.

General form, CPUCOUNT= system option:

CPUCOUNT=*n* |ACTUAL;

where

n

is a number from 1 to 1024 that indicates how many CPUs SAS will assume are available for use by thread-enabled applications.

ACTUAL

causes SAS to detect how many CPUs are available for a specific session.

Caution Setting CPUCOUNT= to a number greater than the actual number of available CPUs might result in reduced overall performance.

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

Note For more information about the CPUCOUNT= system option and other options that are relevant to SAS threading technology, see the SAS documentation.

Calculating and Allocating Sort Resources

Sort Space Requirements

When using PROC SORT to sort an uncompressed data file and replace it with the sorted version, SAS requires enough space in the data library for two copies of the data file that is being sorted as well as additional workspace.

In releases before SAS 9, the workspace required for an uncompressed data file is approximately three to four times the size of the data file. Beginning with SAS 9, the workspace required for an uncompressed data file is approximately twice the size of the data file if the sorted data set is replacing the original. The workspace can be allocated in memory and/or on disk as a utility file, depending on the sort utility and on the options chosen.

You can use the following formula to calculate the amount of work space that the SORT procedure requires:

Formula for calculating the amount of work space needed to sort a SAS data set:

bytes required = (*key-variable-length* + *observation-length*)

** number-of-observations * 4*

where

key-variable-length

is the length of all key variables added together.

observation-length

is the maximum observation length.

number-of-observations

is the number of observations.

Note The multiplier *4* applies only to utility files used in releases before SAS 9 when PROC SORT needs to use disk space in order to sort the data. For in-memory sorting and sorting with SAS 9 and later, the multiplier is *2* or less.

Example

Suppose you want to submit the following program under SAS 9:

```
proc sort data=company.customers;
   by customer_group customer_lastname;
run;
```

You can use the CONTENTS procedure or the DATASETS procedure to obtain the information that is required for the calculation.

Data Set Name	COMPANY.CUSTOMER_DIM	Observations	17157
Member Type	DATA	Variables	11
Engine	V9	Indexes	0
Created	Tuesday, April 26, 2011 03:29:55 PM	Observation Length	208
Last Modified	Tuesday, April 26, 2011 03:29:55 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label			

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

Data Representation	WINDOWS_32		
Encoding	wlatin 1 Western (Windows)		

Figure 23.2: Partial PROC CONTENTS Output

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Format
11	Custcmer_Age	Num	3	
8	Custcmer_Age_group	Char	12	
7	Customer_BirthDate	Num	8	DATE9
2	Customer_Country	Char	2	$COUNTRY.
5	Customer_Firstname	Char	20	
3	Customer_Gender	Char	1	$GENDER.
10	Customer_Group	Char	40	
1	Customer_ID	Num	8	12.
6	Customer_LastName	Char	30	
4	Customer_Name	Char	40	
9	Customer_Type	Char	40	

In this case, the amount of work space needed to sort the data set is 48,575,160 bytes.

```
48,575,160 bytes=(70+200)*89954*2
```

Note The SORT procedure is very I/O intensive. If the file that you are sorting is located in the *Work* library, all of the I/O for the procedure takes place on the file system in which the *Work* library is stored because, by default, the utility files that the SORT procedure creates are created in the *Work* library. Beginning with SAS 9, you can use the UTILLOC= system option to specify one or more file systems in which utility files can be stored. For information about the UTILLOC= option, see the SAS documentation.

Using the SORTSIZE= Option

The SORTSIZE= system option or procedure option specifies how much memory is available to the SORT procedure. Specifying the SORTSIZE=option in the PROC SORT statement temporarily overrides the SAS system option SORTSIZE=

General form, SORTSIZE= option:

SORTSIZE=*memory-specification;*

where

memory-specification

specifies the maximum amount of memory that is available to PROC SORT. Valid values for *memory-specification* are as follows:

MAX

specifies that all available memory can be used.

n

specifies the amount of memory in bytes, where *n* is a real number.

*n*K

specifies the amount of memory in kilobytes, where *n* is a real number.

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

*n*M

specifies the amount of memory in megabytes, where *n* is a real number.

*n*G

specifies the amount of memory in gigabytes, where *n* is a real number.

Note The default value of the SORTSIZE= option depends on your operating environment. See the SAS documentation for your operating environment for more information.

Generally, the value of SORTSIZE= should be less than the physical memory that is available to your process.

If the required workspace is less than or equal to the value specified in the SORTSIZE= system option or procedure option, then the entire sort can take place in memory, which reduces processing time.

If the actual required workspace is greater than the value specified in the SORTSIZE= system option or procedure option, then processing time is increased because the SORT procedure must

1. create temporary utility files in the *Work* directory or mainframe temporary area
2. request memory up to the value specified by SORTSIZE=
3. write a portion of the sorted data to a utility file.

This process is repeated until all of the data is sorted. The SORT procedure then interleaves the data in the utility files to create the final data set.

PROC SORT attempts to adapt to the constraint that is imposed by the SORTSIZE= option. Because PROC SORT uses memory as much as possible,

- a small SORTSIZE=value can increase CPU and I/O resource utilization
- a large SORTSIZE=value can decrease CPU and I/O resource utilization.

Handling Large Data Sets

Dividing a Large Data Set

A data set is too large to sort when there is insufficient room in the data library for a second copy of the data set or when there is insufficient disk space for three to four temporary copies of the data set.

One approach to this situation is to divide the large data set into smaller data sets. The smaller data sets can then be sorted and combined to re-create the large data set. This approach is similar to the process that is used in a threaded sort.

Techniques for dividing and sorting a large data set include

- using PROC SORT with the OUT= statement option and the FIRSTOBS= and OBS= data set options
- using PROC SORT with a WHERE statement
- using subsetting with IF-THEN/ELSE or SELECT-WHEN logic to create multiple output data sets, and sorting the output data sets.

Techniques that can be used to rebuild a large data from smaller, sorted data sets set include

- concatenating the smaller data sets with a SET statement
- interleaving the smaller data sets with SET and BY statements
- appending the smaller data sets with the APPEND procedure.

Comparative Example: Dividing and Sorting a Large Data Set 1

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

Overview

Suppose you want to sort the SAS data set *Retail.Order_fact* by the value of **Order_Date**. The data set is too large to sort using a single SORT procedure. You could accomplish this task by

1. Segmenting by Observation
2. Subsetting Using an IF Statement with the YEAR Function
3. Subsetting Using an IF Statement with a Date Constant
4. Subsetting Using a WHERE Statement with the YEAR Function
5. Subsetting Using a WHERE Statement with a Date Constant.

The following sample programs show each of these techniques. You can use these samples as models for creating benchmark programs in your own environment. Your results might vary depending on the structure of your data, your operating environment, and the resources that are available at your site. You can also view general recommendations for dividing and sorting a large data set.

Programming Techniques

---

❶ Segmentation by Observation

This program segments the data set *Retail.Order_fact* into three smaller data sets by observation number. The three smaller data sets, *Work.One, Work.Two*, and *Work.Three*, are sorted by the value of **Order_Date**. The large data set is then re-created by interleaving the three smaller, sorted data sets.

```
proc sort data=retail.order_fact
          (firstobs = 1 obs = 1500000)
     out=work.one;
   by order_date;
run;

proc sort data=retail.order_fact
          (firstobs = 1500001 obs = 3000000)
      out=work.two;
   by order_date;
run;

proc sort data=retail.order_fact
          (firstobs = 3000001)
     out=work.three;
     by order_date;
run;

data work.orders;
   set work.one work.two work.three;
   by order_date;
run;
```

---

❷ Subsetting Using an IF Statement with the YEAR Function

This program segments the data set *Retail.Order_fact* into three smaller data sets by using a subsetting IF statement and the YEAR function. The three smaller data sets, *Work.One, Work.Two*, and *Work.Three*, are then sorted by the value of **Order_Date**. The large data set is then re-created by concatenating the three smaller, sorted data sets. Interleaving is not required because the smaller data sets do not overlap each other on the sort key **Order_Date**.

data work.one work.two work.three; set retail.order_fact; year=year(order_date); if year in (1998,1999)

```
data work.one work.two work.three;
   set retail.order_fact;
   year=year(order_date);
```


Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

```
   if year in (1998,1999)
      then output work.one;
   else if year in (2000,2001)
      then output work.two;
   else output work.three;
run;

proc sort data=work.one;
   by order_date;
run;

proc sort data=work.two;
   by order_date;
run;
proc sort data=work.three;
   by order_date;
run;

data work.orders;
   set work.one work.two work.three;
run;
```

---

❸ Subsetting Using an IF Statement with a Date Constant

This program segments the data set *Retail.Order_fact* into three smaller data sets by using a subsetting IF statement and a date constant. The three smaller data sets, *Work.One, Work.Two*, and *Work.Three*, are then sorted by the value of **Order_Date**. The large data set is then recreated by concatenating the three smaller, sorted data sets. Interleaving is not required because the smaller data sets do not overlap each other on the sort key **Order_Date**.

```
data work.one work.two work.three;
   set retail.order_fact;
   if order_date <= '31Dec1999'd then
      output work.one;
   else if '31dec1999'd < order_date < '01jan2002'd
      then output work.two;
   else output work.three;
run;

proc sort data=work.one;
   by order_date;
run;

proc sort data=work.two;
   by order_date;
run;

proc sort data=work.three;
   by order_date;
run;

data work.orders;
   set work.one work.two work.three;
run;
```

---

❹ Subsetting Using a WHERE Statement with the YEAR Function

This program segments the data set *Retail.Order_fact* into three smaller data sets by using a WHERE statement and the YEAR function. The three smaller data sets, *Work.One, Work.Two*, and *Work.Three*, are sorted by the value of **Order_Date**. The large data set is then re-created by concatenating the three smaller, sorted data sets. Interleaving is not required because the smaller data sets do not overlap each other on the sort key **Order_Date**.

```
proc sort data=retail.order_fact
     out=work.one;
```


Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

```
   by order_date;
   where year(order_date) in (1998, 1999);
run;

proc sort data=retail.order_fact
     out=work.two;
   by order_date;
   where year(order_date) in (2000, 2001);
run;

proc sort data=retail.order_fact
     out=work.three;
   by order_date;
   where year(order_date) in (2002);
run;

data work.orders;
   set work.one work.two work.three;
run;
```

---

5 Subsetting Using a WHERE Statement with a Date Constant

This program segments the data set *Retail.Order_fact* into three smaller data sets by using a WHERE statement with a date constant. The three smaller data sets, *Work.One, Work.Two*, and *Work.Three*, are sorted by the value of **Order_Date**. The large data set is then re-created by concatenating the three smaller, sorted data sets. Interleaving is not required because the smaller data sets do not overlap each other on the sort key **Order_Date**.

```
proc sort data=retail.order_fact
     out=work.one;
   by order_date;
   where order_date le '31Dec1999'd;
run;

proc sort data=retail.order_fact
     out=work.two;
   by order_date;
   where order_date between '01jan2000'd and
                            '31dec2001'd;
run;

proc sort data=retail.order_fact
     out=wotk.three;
   by order_date;
   gwhere order_date ge '01jan2002'd;
run;

data work.orders;
   set work.one work.two work.three;
run;
```

General Recommendations

- Use a DATA step rather than PROC APPEND to re-create a large data set from smaller subsets.
- Use a constant rather than a SAS function because calling a function repeatedly increases CPU usage.
- Use a subsetting IF with either a constant or a function rather than a WHERE statement with a function.

Comparative Example: Dividing and Sorting a Large Data Set 2

Overview

Like the programs shown in the previous section, each of the following programs illustrates a method for dividing the large data set *Retail.Order_fact* into smaller data sets for sorting. However, in this example, the smaller data sets, *Work.One,*

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

Work.Two, and *Work.Three*, are combined using the APPEND procedure rather than a DATA step in programs 2,3,4, and 5.

1. Segmenting by Observation
2. Subsetting Using an IF Statement with the YEAR Function
3. Subsetting Using an IF Statement with a Date Constant
4. Subsetting Using a WHERE Statement with the YEAR Function
5. Subsetting Using a WHERE Statement with a Date Constant.

The following sample programs show each of these techniques. You can use these samples as models for creating benchmark programs in your own environment. Your results might vary depending on the structure of your data, your operating environment, and the resources that are available at your site. You can also view general recommendations for dividing and sorting a large data set.

Programming Techniques

 Segmenting by Observation

This program segments the data set *Retail.Order_fact* into three smaller data sets by observation number. The three smaller data sets, *Work.One, Work.Two*, and *Work.Three*, are sorted by the value of **`Order_Date`**. PROC APPEND cannot be used to re-create a large data set that was segmented using FIRSTOBS= and OBS=. Therefore, the large data set is re-created by interleaving the three smaller, sorted data sets.

```
proc sort data=retail.order_fact
          (firstobs = 1 obs = 1500000)
     out=work.one;
   by order_date;
run;

proc sort data=retail.order_fact
          (firstobs = 1500001 obs = 3000000)
     out=work.two;
   by order_date;
run;

proc sort data=retail.order_fact
          (firstobs = 3000001)\
     out=work.three;
     by order_date;
run;

data work.orders;
   set work.one work.two work.three;
   by order_date;
run;
```

 Subsetting Using an IF Statement with the YEAR Function

This program segments the data set *Retail.Order_fact* into three smaller data sets by using a subsetting IF statement and the YEAR function. The three smaller data sets, *Work.One, Work.Two*, and *Work.Three*, are then sorted by the value of **`Order_Date`**. The large data set is then re-created using the APPEND procedure.

```
data work.one work.two work.three;
   set retail.order_fact;
   year=year(order_date);
   if year in (1998,1999)
      then output work.one;
   else if year in (2000,2001)
      then output work.two;
```


Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

```
   else output work.three;
run;

proc sort data=work.one;
   by order_date;
run;

proc sort data=work.two;
   by order_date;
run;
proc sort data=work.three;
   by order_date;
run;

proc append base=work.orders data=work.one;
run;
proc append base=work.orders data=work.two;
run;
proc append base=work.orders data=work.three;
run;
```

---

3 Subsetting Using an IF Statement with a Date Constant

This program segments the data set *Retail.Order_fact* into three smaller data sets by using a subsetting IF statement and a date constant. The three smaller data sets, *Work.One, Work.Two*, and *Work.Three*, are then sorted by the value of **Order_Date**. The large data set is then recreated using the APPEND procedure.

```
data work.one work.two work.three;
   set retail.order_fact;
   if order_date <= '31Dec1999'd then
      output work.one;
   else if '31dec1999'd < order_date < '01jan2002'd
      then output work.two;
   else output work.three;
run;

proc sort data=work.one;
   by order_date;
run;

proc sort data=work.two;
   by order_date;
run;

proc sort data=work.three;
   by order_date;
run;

proc append base=work.orders data=work.one;
run;
proc append base=work.orders data=work.two;
run;
proc append base=work.orders data=work.three;
run;
```

---

4 Subsetting Using a WHERE Statement with the YEAR Function

This program segments the data set *Retail.Order_fact* into three smaller data sets by using a WHERE statement and the YEAR function. The three smaller data sets, *Work.One, Work.Two*, and *Work.Three*, are sorted by the value of **Order_Date**. The large data set is then re-created using the APPEND procedure.

```
proc sort data=retail.order_fact
     out=work.one;
```


Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

```
   by order_date;
   where year(order_date) in (1998, 1999);
run;

proc sort data=retail.order_fact
     out=work.two;
   by order_date;
   where year(order_date) in (2000, 2001);
run;

proc sort data=retail.order_fact
     out=work.three;
   by order_date;
   where year(order_date) in (2002);
run;

proc append base=work.orders data=work.one;
run;
proc append base=work.orders data=work.two;
run;
proc append base=work.orders data=work.three;
run;
```

---

5 Subsetting Using a WHERE Statement with a Date Constant

This program segments the data set *Retail.Order_fact* into three smaller data sets by using a WHERE statement with a date constant. The three smaller data sets, *Work.One, Work.Two*, and *Work.Three*, are sorted by the value of **`Order_Date`**. The large data set is then re-created using the APPEND procedure.

```
proc sort data=retail.order_fact
     out=work.one;
   by order_date;
   where order_date le '31Dec1999'd;
run;

proc sort data=retail.order_fact
     out=work.two;
   by order_date;
   where order_date between '01jan2000'd and
                            '31dec2001'd;
run;

proc sort data=retail.order_fact
     out=wotk.three;
   by order_date;
   where order_date ge '01jan2002'd;
run;

proc append base=work.orders data=work.one;
run;
proc append base=work.orders data=work.two;
run;
proc append base=work.orders data=work.three;
run;
```

General Recommendations

- Use a DATA step rather than PROC APPEND to re-create a large data set from smaller subsets.

Using the TAGSORT Option

You can also use the *TAGSORToption* to sort a large data set. The TAGSORT option stores only the BY variables and the observation numbers in temporary tiles. The BY variables and the observation numbers are called *tags*. At the completion of the sorting process, PROC SORT uses the tags to retrieve records from the input data set in sorted order.

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

General form, SORT procedure with the TAGSORT option:

PROC SORT *BATA=SAS-data-set-name* **TAGSORT;**

where

SAS-data-set-name

is a valid SAS data set name.

When the total length of the BY variables is small compared to the record length, TAGSORT reduces temporary disk usage considerably because sorting just the BY variables means sorting much less data. However, processing time is usually higher than if a regular sort is used because TAGSORT increases CPU time and I/O usage in order to save memory and disk space. TAGSORT

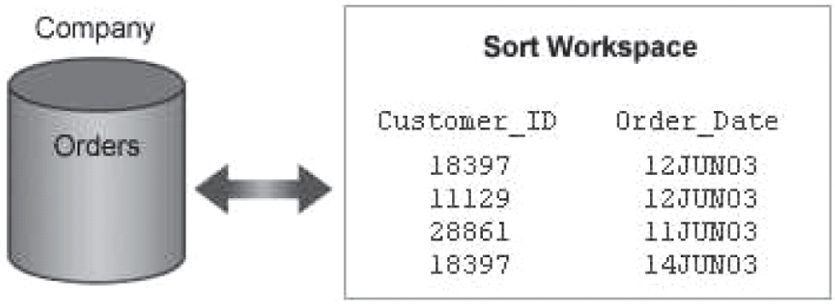
- uses significantly more CPU time and I/O than a regular sort if the data is extremely out of order with regard to the BY variables
- uses slightly more CPU time and I/O than a regular sort if the data is mostly in order with regard to the BY variables.

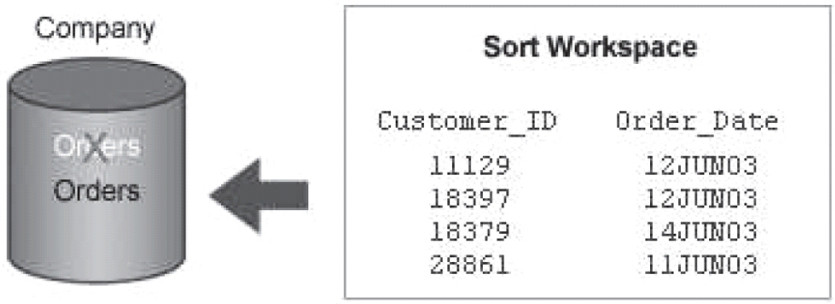
Example

In the following program, only the BY variables, **`Customer_ID`** and **`Order_Date`**, and the tags are stored in temporary files within the sort workspace. SAS then

- sorts the temporary files
- uses the tags to retrieve the observations from the original data set
- re-creates the sorted data set.

```
proc sort data=company.orders tagsort;
   by customer_id order_date;
run;
```





Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

Caution The TAGSORT option is not supported by the threaded sort.

Removing Duplicate Observations Efficiently

Overview

The SORT procedure can be used to remove duplicate observations when it is

- used with the NODUPKEY option
- used with the NODUPRECS option
- followed by FIRST, processing in the DATA step
- used with the DUPOUT= option to write removed observations to a data set.

Generally, PROC SORT with the NODUPKEY option uses less I/O and CPU time than PROC SORT followed by a DATA step that uses FIRST, processing. Before viewing a comparative example, examine each of the techniques that are listed above.

Using the NODUPKEY Option

The NODUPKEY option checks for and eliminates observations that have duplicate BY-variable values. If you specify this option, then PROC SORT compares *all BY-variable values* for each observation to those for the previous observation that was written to the output data set. If an exact match is found, then the observation is not written to the output data set.

General form, PROC SORT with the NODUPKEY option:

PROC SORT *BATA=SAS-data-set-name* **NODUPKEY;**

where

SAS-data-set-name

is a valid SAS data set name.

Example

The SAS data set *Company.Reorder* contains two duplicated observations. Observation 9 is a duplicate of observation 1, and observation 7 is a duplicate of observation 2. The duplicate observations are removed when the data is sorted by the values of **`Product_Line`** and **`Product_Name`** and when the NODUPKEY option is used.

```
proc sort data=company.reorder nodupkey;
   by product_lineproduct_name;
run;
```

Table 23.7: SAS Data Set Company.Reorder

Obs	Product_Line	Product_Name	Supplier_Name
1	Children	Ski Jacket w/Removable Fleece	Scandinavian Clothing A/S
2	Children	Kids Children's Fleece Hat	3Top Sports
3	Clothes & Shoes	Watchit 120 Sterling/Reflective	Eclipse Inc
4	Sports	Sparkle Spray Blue	CrystalClear Optics Inc
5	Outdoors	Money Purse, Black	Top Sports
6	Sports	Mayday Serious Down Jacket	Mayday Inc
7	Children	Kids Children's Fleece Hat	3Top Sports
8	Clothes & Shoes	Tyfoon Linen Pants	Typhoon Clothing
9	Children	Ski Jacket w/Removable Fleece	Scandinavian Clothing A/S

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

Table 23.8: SAS Data Set Company.Reorder, Before Removing Duplicate Observations

Obs	Product_Line	Product_Name	Supplier_Name
1	Children	Kids Children's Fleece Hat	3Top Sports
2	Children	Kids Children's Fleece Hat	3Top Sports
3	Children	Ski Jacket w/Removable Fleece	Scandinavian Clothing A/S
4	Children	Ski Jacket w/Removable Fleece	Scandinavian Clothing A/S
5	Clothes & Shoes	Tyfoon Linen Pants	Typhoon Clothing
6	Clothes & Shoes	Watchit 120 Sterling/Reflective	Eclipse Inc
7	Outdoors	Money Purse, Black	Top Sports
8	Sports	Mayday Serious Down Jacket	Mayday Inc
9	Sports	Sparkle Spray Blue CrystalClear	Optics Inc

Table 23.9: SAS Data Set Company.Reorder, Duplicate Observations Removed

Obs	Product_Line	Product_Name	Supplier_Name
1	Children	Kids Children's Fleece Hat	3Top Sports
2	Children	Ski Jacket w/Removable Fleece	Scandinavian Clothing A/S
3	Clothes & Shoes	Tyfoon Linen Pants	Typhoon Clothing
4	Clothes & Shoes	Watchit 120 Sterling/Reflective	Eclipse Inc
5	Outdoors	Money Purse, Black	Top Sports
6	Sports	Mayday Serious Down Jacket	Mayday Inc
7	Sports	Sparkle Spray Blue CrystalClear	Optics Inc

Using the NODUPRECS Option

The NODUPRECS option cheeks for and eliminates consecutive duplicate observations. Unlike the NODUPKEY option, the NODUPRECS option compares *all of the variable values* for each observation to those for the previous observation that was written to the output data set. If an exact match is found, then the observation is not written to the output data set.

General form, PROC SORT with the NODUPRECS option:

PROC SORT *BATA=SAS-data-set-name* **NODUPRECS;**

where

SAS-data-set-name

is a valid SAS data set name.

Note NODUP is an alias for NODUPRECS.

Because NODUPRECS checks only consecutive observations, some nonconsecutive duplicate observations might remain in the output data set. You can remove all duplicates with this option by sorting on all variables.

Example

When *Company.Reorder* is sorted using the BY variable **`Product_Line`** and the NODUPRECS option, the duplicate observation that contains the product name *Kids Children's Fleece Hat* is eliminated because it exactly matches the observation that was previously written to the output data set. The duplicate observation that contains the product name *Ski Jacket w/Removable Fleece* is retained because it does not exactly match the observation that was previously written

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

to the output data set.

```
proc sort data=company.reorder noduprecs;
   by product_line;
run;
```

Table 23.10: SAS Data Set Company.Reorder

Obs	Product_Line	Product_Name	Supplier_Name
1	Children	Ski Jacket w/Removable Fleece	Scandinavian Clothing A/S
2	Children	Kids Children's Fleece Hat	3Top Sports
3	Clothes & Shoes	Watchit 120 Sterling/Reflective	Eclipse Inc
4	Sports	Sparkle Spray Blue	CrystalClear Optics Inc
5	Outdoors	Money Purse, Black	Top Sports
6	Sports	Mayday Serious Down Jacket	Mayday Inc
7	Children	Kids Children's Fleece Hat	3Top Sports
8	Clothes & Shoes	Tyfoon Linen Pants	Typhoon Clothing
9	Children	Ski Jacket w/Removable Fleece	Scandinavian Clothing A/S

Table 23.11: SAS Data Set Company.Reorder, Before Removing Duplicate Observation

Obs	Product_Line	Product_Name	SupplierName
1	Children	Ski Jacket w/Removable Fleece	Scandinavian Clothing A/S
2	Children	Kids Children's Fleece Hat	3Top Sports
3	Children	Kids Children's Fleece Hat	3Top Sports
4	Children	Ski Jacket w/Removable Fleece	Scandinavian Clothing A/S
5	Clothes & Shoes	Watchit 120 Sterling/Reflective	Eclipse Inc
6	Clothes & Shoes	Tyfoon Linen Pants	Typhoon Clothing
7	Outdoors	Money Purse, Black	Top Sports
8	Sports	Sparkle Spray Blue	CrystalClear Optic Inc
9	Sports	Mayday Serious Down Jacket	Mayday Inc

Table 23.12: SAS Data Set Company.Reorder, Duplicate Observation Remaining

Obs	Product_Line	Product_Name	Supplier_Name
1	Children	Ski Jacket w/Removable Fleece	Scandinavian Clothing A/S
2	Children	Kids Children's Fleece Hat	3Top Sports
3	Children	Ski Jacket w/Removable Fleece	Scandinavian Clothing A/S
4	Clothes & Shoes	Watchit 120 Sterling/Reflective	Eclipse Inc
5	Clothes & Shoes	Tyfoon Linen Pants	Typhoon Clothing
6	Outdoors	Money Purse, Black	Top Sports
7	Sports	Sparkle Spray Blue	CrystalClear Optic Inc
8	Sports	Mayday Serious Down Jacket	Mayday Inc

Both duplicate observations are removed when *Company.Reorder* is sorted by both **`Product_Line`** and **`Product_Name`** and when the NODUPRECS option is used.

```
proc sort data=company.reorder noduprecs;
   by product_lineproduct_name;
run;
```

Table 23.13: SAS Data Set Company.Reorder, Both Duplicate Observations Removed

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

Obs	Product_Line	Product_Name	Supplier_Name
1	Children	Kids Children's Fleece Hat	3Top Sports
2	Children	Ski Jacket w/Removable Fleece	Scandinavian Clothing A/S
3	Clothes & Shoes	Tyfoon Linen Pants	Typhoon Clothing
4	Clothes & Shoes	Watchit 120 Sterling/Reflective	Eclipse Inc
5	Outdoors	Money Purse, Black	Top Sports
6	Sports	Mayday Serious Down Jacket	Mayday Inc
7	Sports	Sparkle Spray Blue CrystalClear	Optics Inc

Note The SORTDUP= system option controls how NODUPRECS processing works. Specifying SORTDUP=PHYSICAL removes duplicates based on all variables in the data set. This is the default. Specifying SORTDUP=LOGICAL removes duplicates based only on the variables that remain after the DROP= and KEEP= data set options are processed. See the SAS documentation for more information.

Using the EQUALS | NOEQUALS Option

EQUALS I NOEQUALS is a SORT procedure option that helps to determine the order of observations in the output data set. When you use NODUPRECS or NODUPKEY to remove observations from the output data set, the choice of EQUALS or NOEQUALS can have an effect on which observations are removed.

EQUALS is the default. For observations that have identical BY-variable values, EQUALS maintains the order from the input data set in the output data set. NOEQUALS does not necessarily preserve this order in the output data set. NOEQUALS can save CPU time and memory resources.

Example

The following program uses PROC SORT with the NODUPKEY option and the NOEQUALS option to create an output data set that contains only the first observation in each BY group. Notice that the output data set *Work.New* contains different observations when the EQUALS option is used.

```
proc sort data=company.products out=work.new
     nodupkey noequals;
   by product_line;
run;
```

Table 23.14: SASDataSet Company.Products

Obs	Product_Line	Product_Name	Suppller_Name
1	Clothes & Shoes	Big Guy Men's Ringer T	Eclipse Inc
2	Children	Boy's and Girl's Ski Pants with Braces	Scandinavian Clothing A/S
3	Outdoors	Cotton Moneybelt/Polyester 45×11	Prime Sports Ltd
4	Sports	Cougar Shorts	SD Sporting Goods Inc
5	Clothes & Shoes	Far Out Teambag S	3Top Sports
6	Children	Kid Basic Tracking Suit	Triple Sportswear Inc
7	Sports	Maxrun Ultra short Sprinter Tights	Force Sports
8	Clothes & Shoes	Wa.leather Street Shoes	Fuller Trading Co.

Table 23.15: SAS Data Set Work.New, NOEQUALS Option Used

Obs	Product_Line	Product_Name	Supplier_Name
1	Children	Kid Basic Tracking Suit	Triple Sportswear Inc
2	Clothes & Shoes	Far Out Teambag S	3Top Sports
3	Outdoors	Cotton Moneybelt/Polyester 45×11	Prime Sports Ltd
4	Sports	Maxrun Ultra short Sprinter Tights	Force Sports

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

Table 23.16: SAS Data Set Work.New, EQUALS Option Used

Obs	Product_Line	Product_Name	Supplier_Name
1	Children	Boy's and Girl's Ski Pants with Braces	Scandinavian Clothing A/S
2	Clothes & Shoes	Big Guy Men's Ringer T	Eclipse Inc
3	Outdoors	Cotton Moneybelt/Polyester 45×11	Prime Sports Ltd
4	Sports	Cougar Shorts	SD Sporting Goods Inc

Note The EQUALS | NOEQUALS option is supported by the threaded sort. However, I/O performance might be reduced when you use the EQUALS option because partitioned data sets will be processed as if they are non-partitioned data sets.

Caution The order of observations within BY groups that are returned by the threaded sort might not be consistent between runs. Therefore, using the NOEQUALS option can produce inconsistent results in your output data sets.

Using FIRST. LAST. Processing in the DATA Step

FIRST. LAST, processing in the DATA step can also be used to remove duplicate observations from a SAS data set.

In the data set *Company.Onorder*, the fourth observation contains a duplicate value for **`Product_Name`**. The following program removes the observation that contains the duplicate value by first sorting the input data set, *Company.Onorder*, by the value of **`Product_Name`**. The DATA step then selects only the first observation in the BY group.

```
proc sort data=company.onorder
          out=work.sorted;
   by product_name;
run;
data work.onorder2;
   set work.sorted;
   by product_name;
   if first.product_name;
run;
```

Table 23.17: SAS Data Set Company.Onorder

Obs	Product_Line	Product_Name	Quantity
1	Clothes & Shoes	Big Guy Men's Ringer T	70
2	Children	Boy's and Girl's Ski Pants with Braces	55
3	Outdoors	Cotton Moneybelt/Polyester 45×11	20
4	Sports	Big Guy Men's Ringer T	70
5	Sports	Cougar Shorts	40
6	Clothes & Shoes	Far Out Teambag S	32
7	Children	Kid's Basic Tracking Suit	20
8	Sports	Maxrun Ultra short Sprinter Tights	25
9	Clothes & Shoes	Wa.leather Street Shoes	30

Table 23.18: SAS Data Set Company.Onorder2

Obs	Product_Line	Product_Name	Quantity
1	Clothes & Shoes	Big Guy Men's Ringer T	70
2	Children	Boy's and Girl's Ski Pants with Braces	55
3	Outdoors	Cotton Moneybelt/Polyester 45×11	20
4	Sports	Cougar Shorts	40
5	Clothes & Shoes	Far Out Teambag S	32
6	Children	Kid's Basic Tracking Suit	20

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

7	Sports	Maxrun Ultra short Sprinter Tights	25
8	Clothes & Shoes	Wa.leather Street Shoes	30

Comparative Example: Removing Duplicate Observations Efficiently

Overview

Suppose you want to remove observations from the data set *Retail.Order_fact* in which the value of **`Order_Date`** is duplicated. Among the techniques you could use are

1. The NODUPKEY Option and the EQUALS Option
2. The NODUPKEY Option and the NOEQUALS Option
3. PROC SORT and a DATA Step with BY-Group and FIRST. Processing

The following sample programs show each of these techniques. You can use these samples as models for creating benchmark programs in your own environment. Your results might vary depending on the structure of your data, your operating environment, and the resources that are available at your site. You can also view general recommendation for eliminating duplicates.

Programming Techniques

---

❶ The NODUPKEY Option and the EQUALS Option

This program uses the NODUPKEY option and the EQUALS option to check for and eliminate observations that have duplicate BY-variable values. For observations that have identical BY-variable values, the EQUALS option maintains the order from the input data set in the output data set.

```
proc sort data=retail.order_fact
          out=work.sorted
   nodupkey equals;
   by order_date;
run;
```

---

❷ The NODUPKEY Option and the NOEQUALS Option

This program uses the NODUPKEY option and the NOEQUALS option to check for and eliminate observations that have duplicate BY-variable values. For observations that have identical BY-variable values, the NOEQUALS option does not necessarily maintain the order from the input data set in the output data set, which can save CPU time and memory.

```
proc sort data=retail.order_fact
     out=work.sorted
     nodupkey noequals;
   by order_date;
run;
```

---

❸ PROC SORT and a DATA Step with BY-Group and FIRST. Processing

In this program, the input data set is first sorted using the SORT procedure. Duplicate observations are then removed using BY-group and FIRST, processing.

```
proc sort data=retail.order_fact
          out=work.sorted;
   by order_date;
run;
data work.sorted;
   set work.sorted;
   by order_date;
```


Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

```
   if first.order_date;
run;
```

Comparing Techniques to Eliminate Duplicate Data

The following table compares the techniques for eliminating duplicate data using PROC SORT, DATA step, and PROC SQL:

Table 23.19: Comparing Techniques for Eliminating Duplicate Data

Technique	Advantages	Disadvantages
PROC SORT with NODUPKEY and DUPOUT=	▪ No additional passes of the data occur. ▪ Only the PROC SORT step is required. ▪ Unique observations should be placed in one data set and other observations in another data set.	You cannot specify where observations are output
DATA step with `IF First.by-var=1 and LAST.by-var=1;`	▪ The DATA step has many capabilities for further data manipulation. ▪ Unique observations should be placed in one data set and other observations in another data set.	Two steps are required.
DATA step with `IF FIRST.by-var=1;`	The DATA step has many capabilities for further data manipulation.	Two steps are required.
PROC SQL with the SELECT DISTINCT statement	The SQL step has many capabilities for combining, ordering, and grouping data.	▪ The DISTINCT keyword applies to all of the variables in the SELECT statement. ▪ Only one data set can be created.

General Recommendations

- To remove duplicate observations from a SAS data set, use PROC SORT with the NODUPKEY option rather than a PROC SORT step followed by a DATA step that uses FIRST, processing.
- Be careful not to confuse NODUPKEY with NODUPRECS. NODUPRECS compares variables in the observations.

Additional Features

Selecting a Host Sort Utility

Host sort utilities are third-party sort packages that are available in some operating environments. In some cases, using a host sort utility with PROC SORT might be more efficient than using the SAS sort utility with PROC SORT.

The following table lists the host sort utilities that might be available at your site. SAS uses the values that are set for the SORTPGM=, SORTCUT=, SORTCUTP=, and SORTNAME= system options to determine which sort to use.

Operating Environment	Host Sort Utilities
z/OS	Dfsort (default)Syncsort
UNIX	CosortSyncsort (default)
Windows	Syncsort

Note Ask your system administrator whether a host sort utility is available at your site. For more information about host sort utilities, see the SAS documentation for your operating environment.

Using the SORTPGM= System Option

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

The value specified in the SORTPGM= system option tells SAS whether to use the SAS sort, to use the host sort, or to determine which sort utility is best for the data set.

General form, SORTPGM= system option:

OPTIONS SORTPGM= BEST | HOST | SAS;

where

BEST

specifies that SAS chooses the sort utility. This is the default.

HOST

specifies that the host sort utility is always used.

SAS

specifies that the SAS sort utility is always used.

Using the SORTCUTP= System Option

The SORTCUTP= system option specifies the *number of bytes* above which the host sort utility is used instead of the SAS sort utility.

General form, SORTCUTP= system option:

OPTIONS SORTCUTP= *n*/ *n*K/ *n*M/ *n*G/ MIN / MAX / *hexX;*

where

n/ *n*K/ *n*M/ *n*G/

specifies the value in bytes, kilobytes, megabytes, or gigabytes, respectively.

MIN

specifies the minimum value.

MAX

specifies the maximum value.

hexX

specifies the value as a hexadecimal number of bytes.

Note To determine the minimum and maximum values for SORTCUTP=, see the SAS documentation for your operating environment.

The following table lists the default values for SORTCUTP= in the z/OS, UNIX, and Windows operating environments.

Operating Environment	Default SORTCUTP= Value	Default Behavior
z/OS	4M	SAS sort is used until this value is reached
UNIX	0	SAS sort is always used
Windows	0	SAS sort is always used

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

Using the SORTCUT= System Option

The SORTCUT= system option can be used to specify the *number of observations* above which the host sort utility is used instead of the SAS sort utility.

Note The SORTCUT= system option is not available in the z/OS operating environment.

General form, SORTCUT= system option:

OPTIONS SORTCUT=*n*| *n*K| *n*M| *n*G | MIN | MAX | *hexX;*

where

n| *n*K| *n*M| *n*G

specifies the number of observations.

MIN

specifies *0* observations.

MAX

specifies the maximum number of observations.

hexX

specifies the number of observations in hexadecimal notation.

Note To determine the maximum value for SORTCUT=, see the SAS documentation for your operating environment.

The default value of the SORTCUT= system option is *0*.

Using the SORTNAME= System Option

The SORTNAME= option specifies the host sort utility that will be used if the value of SORTPGM= is *BEST* or *HOST*.

Note The SORTNAME= system option is not available in the Windows operating environment.

General form, SORTNAME= option:

OPTIONS SOKTNAME=*host-sort-utility name;*

where

host-sort-utility name

is the name of a valid sort host utility.

Example

When you specify SORTPGM=BEST, SAS uses the value of the SORTCUT= and SORTCUTP= options to determine whether to use the host sort or the SAS sort. If you specify values for both the SORTCUT= and SORTCUTP= options, and if either condition is true, SAS chooses the host sort.

In the program below, if the size of the SAS data set *Company.Orders* is larger than 10,000 bytes, the host sort utility, Syncsort, will be used instead of the SAS sort utility.

```
options sortpgm=best sortcutp=10000
sortname=syncsort;
```


Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

```
proc sort data=company.orders out=company.deliveries;
   by delivery_date;
run;
```

Summary

Avoiding Unnecessary Sorts

When BY-group processing with an index is used, the data can be sequenced by different variables without having to repeat the SORT procedure ifmultiple indexes are used. Because indexes are updated automatically, there is no need to re-sort a data set when observations are modified or added. However, BY-group processing with an index is less efficient than reading a sorted data set sequentially, and storage space is required for the index.

You can also use the NOTSORTED option with a BY statement to create ordered or grouped reports without sorting the data. The NOTSORTED option specifies that observations that have the same BY value are grouped together but are not necessarily sorted in alphabetical or numeric order. The NOTSORTED option works best when observations that have the same BY value are stored together.

The NOTSORTED option can be used with FIRST, and LAST., which are temporary automatic variables in the PDV that identify the first and last observations in each BY group. These temporary variables are available for DATA step programming but are not added to the output data set.

The GROUPFORMAT option is useful when you have defined formats for grouped data. The GROUPFORMAT option uses the formatted values of avariable, instead of the internal values to determine where a BY group begins and ends, and how FIRST, and LAST, are computed. When the GROUPFORMAT option is used, the data set must be sorted by the GROUPFORMAT variable or grouped by the formatted values of the GROUPFORMAT variable.

You can use a CLASS statement to specify the variables whose values define the subgroup combinations for an analysis by a SAS procedure. Unlike the BY statement, when the CLASS statement is used with Base SAS procedures, it does not require the data to be presorted using the BY-variable values or that you have an index based on the BY variables.

If you are working with input data that is already sorted, you can specify how the data is ordered by using the SORTEDBY= data set option. Although the SORTEDBY= option does not sort a data set, it sets the Sorted indicator on the data set.

Review the related comparative examples:

- "Comparative Example: Using BY-Group Processing with an Index to Avoid a Sort" onpage 813
- "Comparative Example: Using a BY or CLASS Statement to Avoid a Sort" on page 821.

Using a Threaded Sort

Beginning with SAS 9, the SORT procedure can take advantage of threaded processing. Threadedjobs are completed in substantially less real time than if each task is handled sequentially. However, the CPU time for threadedjobs is generally increased.

Threaded sorting is enabled or disabled by using the THREADS | NOTHREADS SAS system option or procedure option. The procedure option overrides the value of the system option.

When a threaded sort is used, the observations in the input data set are divided into equal temporary subsets, based on how many processors are allocated to the SORT procedure. Each subset is then sorted on a different processor. The sorted subsets are then interleaved to re-create the sorted version of the input data set.

The performance of athreaded sort is affected by the value of the CPUCOUNT= system option. CPUCOUNT= specifies the number of processors that thread-enabled applications should assume will be available for concurrent processing. SAS uses this information to determine how many threads to start, not to restrict the number of CPUs that will be used.

Calculating and Allocating Sort Resources

When data is sorted, SAS requires enough space in the data library for two copies of the data file that is being sorted, as well as additional workspace.

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

In releases before SAS 9, the required workspace is approximately three to four times the size of the data file. Beginning with SAS 9, the required workspace is approximately twice the size of the data file. The workspace can be allocated in memory and/or on disk as a utility file, depending on which sort utility and options are specified.

The SORTSIZE= option specifies how much memory is available to the SORT procedure. Generally, the value of SORTSIZE= should be less than the physical memory that is available to your process. If the required workspace is less than or equal to the value specified in the SORTSIZE= system option or procedure option, then the entire sort can take place in memory, which reduces processing time.

Handling Large Data Sets

A data set is too large to sort when there is insufficient room in the data library for a second copy of the data set or when there is insufficient disk space for three to four temporary copies of the data set.

One approach to this situation is to divide the large data set into smaller subsets. The subsets can then be sorted and combined to re-create the large data set.

You can also use the TAGSORT option to sort a large data set. The TAGSORT option stores only the BY variables and the observation numbers in temporary files. The BY variables and the observation numbers are called tags. At the completion of the sorting process, PROC SORT uses the tags to retrieve records from the input data set in sorted order.

When the total length of the BY variables is small compared to the record length, TAGSORT reduces temporary disk usage considerably because sorting just the BY variables means sorting much less data. However, processing time might be much higher because the TAGSORT option increases CPU and I/O usage in order to save memory and disk space.

Review the related comparative examples:

- "Comparative Example: Dividing and Sorting a Large Data Set 1" on page 830
- "Comparative Example: Dividing and Sorting a Large Data Set 2" on page 835.

Removing Duplicate Observations Efficiently

The NODUPKEY option cheeks for and eliminates observations that have duplicate BY-variable values. If you specify this option, then PROC SORT compares all BY-variable values for each observation to those for the previous observation that was written to the output data set. If an exact match is found, then the observation is not written to the output data set.

The NODUPRECS option checks for and eliminates duplicate consecutive observations. However, unlike the NODUPKEY option, the NODUPRECS option compares all of the variable values for each observation to those for the previous observation that was written to the output data set. If an exact match is found, then the observation is not written to the output data set.

EQUALS I NOEQUALS is a procedure option that helps to determine the order of observations in the output data set. When you use NODUPRECS or NODUPKEY to remove observations from the output data set, the choice of EQUALS or NOEQUALS can have an effect on which observations are removed.

EQUALS is the default. For observations that have identical BY-variable values, EQUALS maintains the order from the input data set in the output data set. NOEQUALS does not necessarily preserve this order in the output data set. NOEQUALS can save CPU time and memory resources.

FIRST. LAST, processing in the DATA step can also be used to remove duplicate observations in a SAS data set.

Review the related comparative example:

- "Removing Duplicate Observations Efficiently" on page 841.

Additional Features

Depending on your operating environment, you might be able to use additional sorting options, called host sort utilities. Host sort utilities are third-party sort packages. In some cases, using a host sort utility might be more efficient than using the SAS sort utility with PROC SORT.

SAS uses the values that are set for the SORTPGM=, SORTCUTP=, SORTCUT=, and SORTNAME= system options to

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

determine which sort utility to use.

Quiz

Select the best answer for each question. After completing the quiz, check your answers using the answer key in the appendix.

1. When the following program is submitted, what is the value of **`FIRST.Product_Line`** for the third observation in the data set *Work.Products?*

```
data new.products;
   set work.products
   by product_line notsorted;
run;
```

Table 23.20: SAS Data Set Work.Products

Obs	Product_Line	Product_Name	Supplier_Name
1	Children	Kids Children's Fleece Hat	3Top Sports
2	Children	Ski Jacket w/Removable Fleece	Scandinavian Clothing A/S
3	Clothes & Shoes	Tyfoon Linen Pants	Typhoon Clothing
4	Clothes & Shoes	Watchit 120 Sterling/ Reflective	Eclipse Inc
5	Clothes & Shoes	Money Belt, Black	Top Sports

   a. *1*
   b. *3*
   c. *0*
   d. *clothes & Shoes*

2. Which option is used with the SORT procedure to store only the BY variables and the observation numbers in temporary files?
   a. NOTSORTED
   b. GROUPFORMAT
   c. TAGSORT
   d. SORTEDBY=

3. Which of the following is not an advantage of BY-group processing with an index that is based on the BY variables?
   a. The data can be sequenced without using the SORT procedure.
   b. There is no need to re-sort a data set when observations are modified or added.
   c. It is generally more efficient than reading a sorted data set sequentially.
   d. The data can be sequenced by different variables ifmultiple indexes are used.

4. Which SORT procedure option compares all of the variable values for each observation to those for the previous observation that was written to the output data set?
   a. NODUPKEY
   b. NODUPRECS
   c. EQUALS
   d. NOEQUALS

5. What happens if the workspace that is required for completing a sort is less than or equal to the value that is specified in the SORTSIZE= system option or procedure option?
   a. CPU time is increased.

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

b. I/O is increased.

c. The entire sort can take place in memory.

d. A temporary utility file is created in the *Work* directory or in a mainframe temporary area.

Answers

1. Correct answer: a

`First`. is a temporary automatic variable that identifies the first observation in each BY group. When an observation is the first in a BY group, SAS sets the value of the **`FIRST`**, *`variable`* to *1*. For all other observations in the BY group, the value of the **`FIRST`**. *`variable`* is *0*.

2. Correct answer: c

The TAGSORT option stores only the BY variables and the observation numbers in temporary tiles. The BY variables and the observation numbers are called tags. At the completion of the sorting process, PROC SORT uses the tags to retrieve records from the input data set in sorted order.

3. Correct answer: c

When BY-group processing is used with an index that is based on the BY variables, the data can be sequenced without using the SORT procedure. The data can be sequenced by different variables ifmultiple indexes are used. Because indexes are updated automatically, there is no need to re-sort a data set when observations are modified or added. However, BY-group processing with an index is generally less efficient than reading a sorted data set sequentially.

4. Correct answer: b

The NODUPRECS option compares all of the variable values for each observation to those for the previous observation that was written to the output data set. If an exact match is found, then the observation is not written to the output data set.

5. Correct answer: c

The SORTSIZE= system option or procedure option specifies how much memory is available to the SORT procedure. If the required workspace is less than or equal to the value specified in the SORTSIZE= system option or procedure option, then the entire sort can take place in memory, which reduces processing time.

Reprinted for OET7P/639280, Accenture SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited